تکلیف شماره 3

شماره گروه: 8

اعضاي گروه: آرمين افتخاري(9622762033)، محمد رضا پوررضا (9612762592)، رضا برزگر طرقبه (9622762384)،

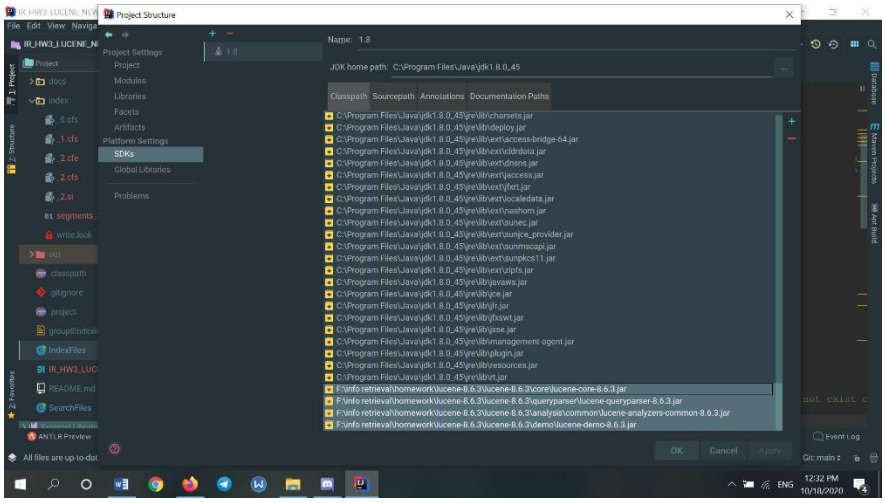
سبحان مرادیان(9622762066) و محمد سلیمان بهزاد(9622762453)

لينك "github": https://github.com/IRGroup8/IR_HW3_LUCENE_NEW.git

قسمت اول – نصب و راه اندازی لوسین

1. فایل زیپ مربوطه به آخرین ورژن آپاچی لوسین را دانلود میکنیم

2. بعد از ایجاد پروژه در محیط برنامه نویسی جاوا (IDE) در بخش classpath محیط، 4 فایل jar زیر را از دایرکتوری های core,analyzer common , demo, queryparser اضافه میکنیم:



قسمت دوم – ساختار پروژه

داکیومنت های مربوط به گروه را جداسازی کرده و در فایل های txt به صورت جداگانه در داخل دایرکتوری docs قرار داده ایم.

```
from nltk.corpus import stopwords
    from nltk.tokenize import word_tokenize
    from nltk.stem.porter import *
    import json

    file = open("group8Indexing.txt", "r")
    text = file.read()
    list = []
    docs = text.split(".I")
    stop_words = set(stopwords.words('english'))
    stemmer = PorterStemmer()
    for doc in docs:
    document = doc.split(".W")
    if (len(document) > 1):
    docNum = document[0]
    docText = document[1]
    with open(docNum.rstrip()+".txt", "w") as output:
    output.write(docText)
```

کد مربوط به جداسازی داکیومنت ها

کلاس IndexFiles مربوط به index کردن داکیومنت هاست و فایل های خروجی index شده را در داخل دایرکتوری index قرار میدهیم

کلاس SearchFiles مربوط به جستجوی کوئری مربوطه در ایندکس ها می باشد.

قسمت سوم indexFiles –

این کلاس شامل سه تابع اصلی است.

- تابع main
- تابع indexDocs
- تابع indexDoc

تابع **main**

```
String indexPath = "index";
String docsPath = "docs";
```

رشته های indexPath و docsPath که به ترتیب مسیرهای مربوط به ذخیره سازی ایندکس ها و داکیومنت ها را نشان می دهند.

```
if (docsPath == null) {
    System.err.println("docs Path is null!");
    System.exit(1);
}

final Path docDir = Paths.get(docsPath);
if (!Files.isReadable(docDir)) {
    System.out.println("Document directory '"
+docDir.toAbsolutePath()+ "' does not exist or is not readable, please
check the path");
    System.exit(1);
}
```

در دو if بعدی چک می شود که داکیومنت وجود داشته و قابل خواندن باشد. لازم به ذکر است که مقدار docsPath باید به شکل آبجکتی از کلاس Path باشد.

```
System.out.println("Indexing Started !");
Directory dir = FSDirectory.open(Paths.get(indexPath));
Analyzer analyzer = new StandardAnalyzer();
IndexWriterConfig iwc = new IndexWriterConfig(analyzer);
iwc.setOpenMode(IndexWriterConfig.OpenMode.CREATE);
```

```
IndexWriter writer = new IndexWriter(dir, iwc);
indexDocs(writer, docDir);
writer.close();
```

حال فرآیند indexing را شروع میکنیم. در ابتدا با استفاده از FSDIRECTORY مسیر مربوط به ایندکس ها را داخل یک دایرکتوری قرار می دهیم. آبجکتی از کلاس StandardAnalyzer تعریف می کنیم تا بوسیله آن، بتوان نرمال سازی را روی ایندکس ها انجام داد.

سپس از این آنالایزر برای کانفیگ کردن IndexWriter استفاده می کنیم و مود آن را روی حالت Create قرار میدهیم.

حال آبجکتی از IndexWriter داریم که آدرس دایرکتوری و کانفیگ انجام شده را به آن پاس میدهیم.

حال تابع IndexDocs را فراخوانی می کنیم که با استفاده از این indexWriterتعریف شده و همینطور دایرکتوری مربوط به داکیومنت ها، عمل indexing را انجام میدهد.

در پایان، writer را close میکنیم.

تابع **indexDocs**

```
public static void indexDocs(final IndexWriter writer, Path path)
throws IOException {
    Files.walkFileTree(path, new SimpleFileVisitor<Path>() {
        @Override
        public FileVisitResult visitFile(Path file,
BasicFileAttributes attrs) throws IOException {
            try {
                indexDoc(writer, file); } catch (IOException e) {
                    e.printStackTrace();
                }
                return FileVisitResult.CONTINUE;
        }
    });
}
```

این تابع بدین شکل عمل میکند که به وسیله walkFileTree به صورت بازگشتی روی تمام فایل های موجود در یک دایرکتوری پیمایش کرده و عمل indexing را با استفاده از تابع **indexDoc** روی هرکدام انجام میدهد. و در با برگرداندن Continue به سراغ فایل بعدی می رود.

تابع **indexDoc**

```
public static void indexDoc(IndexWriter writer, Path file) throws
IOException {
    try (InputStream stream = Files.newInputStream(file)) {
        Document doc = new Document();
        Field pathField = new StringField("path", file.toString(),
```

```
Field.Store.YES);
        doc.add(pathField);
        doc.add(new TextField("contents", new BufferedReader(new
InputStreamReader(stream, StandardCharsets.UTF_8))));
        System.out.println("adding " + file);
        writer.addDocument(doc);
    }
    catch (IOException e){
        e.printStackTrace();
    }
}
```

این تابع وظیفه ایندکس کردن را برعهده دارد و بخش اصلی این کلاس به حساب می آید. در این تابع یک شی از جنس داکیومنت ایجاد می کنیم و سپس فیلد های مختلف سند شامل : مسیر و محتوا را به آن اضافه میکنیم و محتوای موجود در داکیومنت را با استفاده از InputStream داخل شی doc قرار می دهیم. در انتها نیز با استفاده از indexWriter ورودی، ذخیره سازی داکیومنت را انجام میدهیم.

بدین ترتیب، اسناد موجود، ایندکس شده اند و آماده عملیات Search هستند.

بخش سوم : search files class

این کلاس شامل دو تابع اصلی است.

تابع main():

- در ابتدا سه متغیر تعریف میکنیم که متغیر اول شامل آدرس محل ذخیره سازي نمایه ها است. متغیر دوم ناحیه اي است است که میخواهیم عمل جست‌وجو را در آن ناحیه انجام دهیم و متغیر سوم تعداد موارد یافت شده در هر صفحه از خروجي است.
- یك شئ از کلاس indexReader تعریف می‌کنیم و با استفاده از کلاس Path،FSDirectory مربوط به فایل های نمایه‌گذاری شده را باز میکنیم.
- یک شئ از کلاسindexSearcher ایجاد میکنیم که محتوای خوانده شده توسط شئ indexReader به عنوان پارامتر ورودی دریافت میکند.
- یک شئ از کلاس analyser ایجاد میکنیم تا پیش‌پردازش های لازم روی کوئری را انجام دهیم. (analyser استفاده شده در این مرحله باید متابق با analyser استفاده شده در مرحله نمایه‌گذاری باشد. Analyser مورد استفاده ما در این پروژه standardAnalyser میباشد که از الگوریتم porter برای ریشه‌یابی استفاده میکند و stopWord ها را حذف میکند و توکن سازی نیز انجام میدهد)
- یک شئ از کلاس bufferReader ایجاد میکنیم تا بتوانیم ورودی را از کاربر دریافت کنیم.
- یک شئ از کلاس queryParser ایجاد میکنیم و به عنوان پارامتر به constructor این کلاس شئ field و analyser که قبلا تعریف کرده ایم را میدهیم.

```
String index = "index";
String field = "contents";
int hitsPerPage = 10;
IndexReader reader =
DirectoryReader.open(FSDirectory.open(Paths.get(index)));
IndexSearcher searcher = new IndexSearcher(reader);
Analyzer analyzer = new StandardAnalyzer();
BufferedReader in = null;
in = new BufferedReader(new InputStreamReader(System.in,
StandardCharsets.UTF_8));
QueryParser parser = new QueryParser(field, analyzer);
```

- در ادامه در یک حلقه بی‌نهایت از کاربر کوئری مورد نظر را دریافت میکنیم و درستی آن را چک میکنیم.
- یک شئ از کلاس کوئری ایجاد میکنیم و با استفاده از شئ ایجاد شده از کلاس parser کوئری کاربر را parse میکنیم.
- تابع doPagingSearch با آرگومان های input،searcher، query و متغیر hitsPerPage صدا میزنیم
- در پایان حلقه، برای جلوگیری از رخداد خطا، شئ ایجاد شده از کلاس reader را close میکنیم.

```
while (true) {
    System.out.println("Enter query: ");
    String line = in.readLine();
    if (line == null || line.length() == -1) {
        break;
    }
    line = line.trim();
    if (line.length() == 0) {
        break;
    }
```

```
        Query query = parser.parse(line);
        System.out.println("Searching for: " +
    query.toString(field));
        doPagingSearch(in, searcher, query, hitsPerPage);
    }
    reader.close();
```

تابع **doPagingSearch** :

در این تابع :

- با استفاده از تابع search که در کلاس searcher تعریف شده است و به عنوان پارامتر ورودی شئ کوئری و یک عدد برای سقف تعداد نتایج یافت شده برای جست‌وجو، میگیرد.
- scoreDocs و totalHits را در قالب یک شئ topDocs برمیگردانیم.
- قسمت scoredocs خروجی دستور بالا را در یک آرایه از کلاس scoreDocs قرار میدهیم.
- یک متغیر برای ذخیره سازی کل موارد یافت شده ایجاد میکنیم.

```
TopDocs results = searcher.search(query, 20 * hitsPerPage);
ScoreDoc[] hits = results.scoreDocs;
```

تعامل با کاربر

```
while (true) {
        for (int i = start; i < end; i++) {
            Document doc = searcher.doc(hits[i].doc);
            String path = doc.get("path");
            if(path!=null) {
                System.out.println("PATH : " + path + " docID=" +
hits[i].doc + " score=" + hits[i].score);
                continue;
            }
            else{
                System.out.println("NO PATH FOUND");
            }
        }
        if (end == 0) {
            break;

        }
```

```
        if (numTotalHits >= end) {
            boolean quit = false;
            while (true) {
                System.out.print("Press ");
                if (start - hitsPerPage >= 0) {
                    System.out.print("(p)revious page, ");

                }
                if (start + hitsPerPage < numTotalHits) {
                    System.out.print("(n)ext page, ");

                }
                System.out.println("(q)uit or enter number to jump to
a page.");

                String line = in.readLine();
                if (line.length() == 0 || line.charAt(0) == 'q') {
                    quit = true;
                    break;

                }
                if (line.charAt(0) == 'p') {
                    start = Math.max(0, start - hitsPerPage);
                    break;

                } else if (line.charAt(0) == 'n') {
                    if (start + hitsPerPage < numTotalHits) {
                        start += hitsPerPage;

                    }
                    break;

                } else {
                    int page = Integer.parseInt(line);
                    if ((page - 1) * hitsPerPage < numTotalHits) {
                        start = (page - 1) * hitsPerPage;
                        break;
                    } else {
                        System.out.println("No such page");
                    }
                }
            }
            if (quit) break;
            end = Math.min(numTotalHits, start + hitsPerPage);
        }
    }
}
```

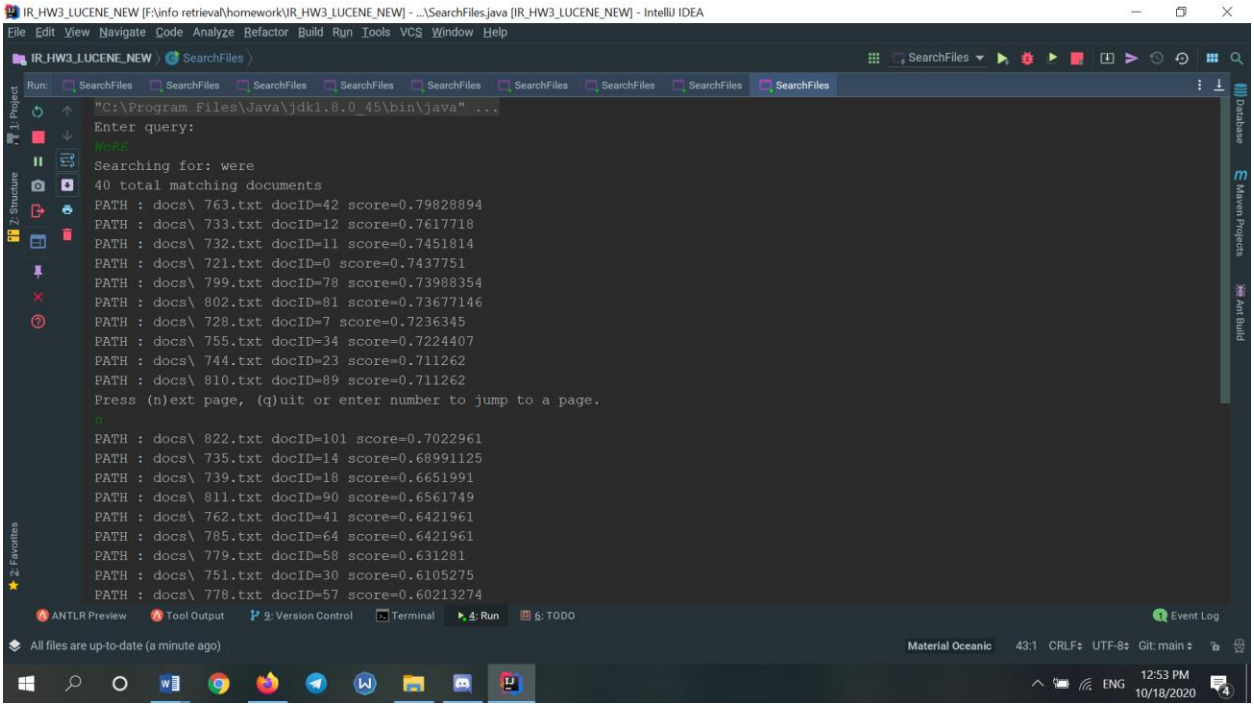
ابتدا با ران کردن فایل IndexFiles ایندکس های مربوطه در دایرکتوری index ذخیره می شود.

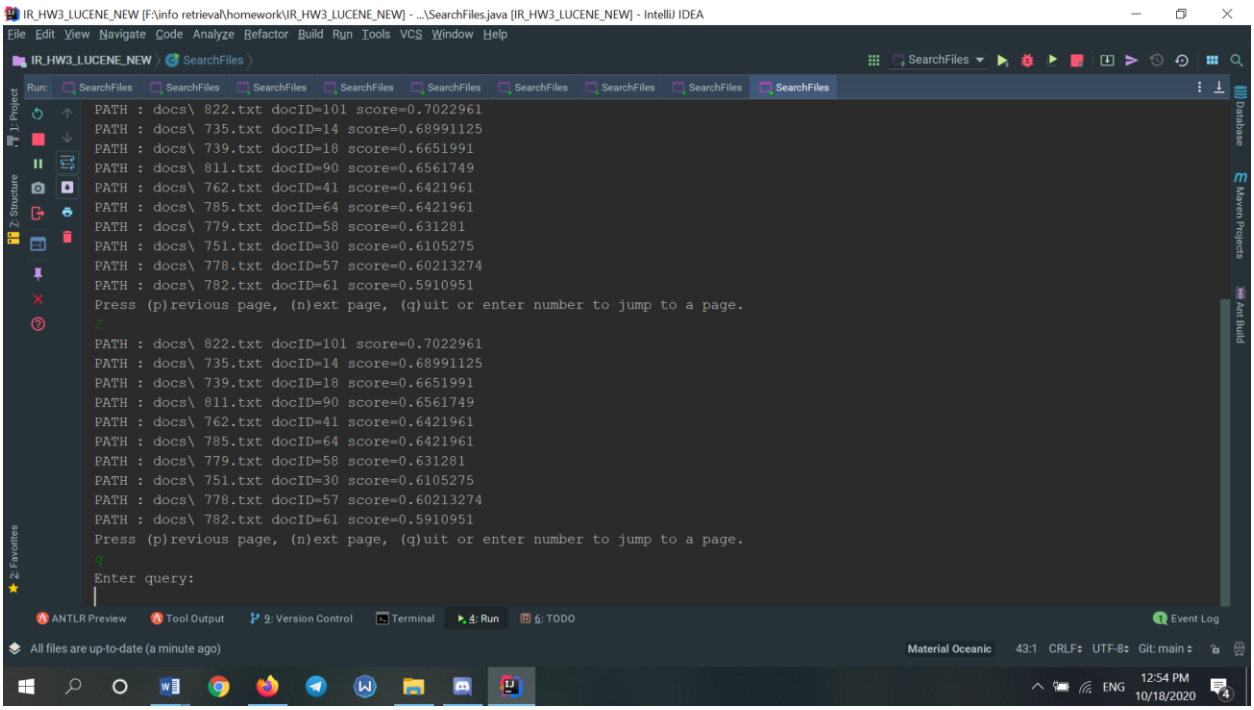
حال با اجرای SearchFiles وارد پروسه Search می شویم.

در ابتدا از کاربر تقاضا می شود که کوئری مورد نظر خود جهت جستجو را وارد کند. در خروجی شکل Normalize شده را نمایش می دهد و سپس در خط بعدی تعداد کل سندهای یافت شده که شامل کوئری مورد نظر هستند را نمایش می دهد. نتایج یافت شده در قالب صفحاتی به کاربر نمایش داده می شود و می تواند با استفاده از کلید های N و P بین صفحات جابه جا شود. (اولویت نمایش، برحسب وزنی است که مقدار آن با استفاده از الگوریتم tf.idf بدست می آید).

$$score(q,d) = coord\text{-}factor(q,d) \cdot query\text{-}boost(q) \cdot \frac{V(q) \cdot V(d)}{|V(q)|} \cdot doc\text{-}len\text{-}norm(d) \cdot doc\text{-}boost(d)$$

برای انجام جستجوی بعدی، از کاراکتر q و یا برای وارد شدن به صفحه بعدی شماره صفحه را می زنیم.


IR_HW3_LUCENE_NEW [F:\info retrieval\homework\IR_HW3_LUCENE_NEW] - ...\SearchFiles.java [IR_HW3_LUCENE_NEW] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
"C:\Program Files\Java\jdk1.8.0_45\bin\java" ...
Enter query:
were
Searching for: were
40 total matching documents
PATH : docs\ 763.txt docID=42 score=0.79828894
PATH : docs\ 733.txt docID=12 score=0.7617718
PATH : docs\ 732.txt docID=11 score=0.7451814
PATH : docs\ 721.txt docID=0 score=0.7437751
PATH : docs\ 799.txt docID=78 score=0.73988354
PATH : docs\ 802.txt docID=81 score=0.73677146
PATH : docs\ 728.txt docID=7 score=0.7236345
PATH : docs\ 755.txt docID=34 score=0.7224407
PATH : docs\ 744.txt docID=23 score=0.711262
PATH : docs\ 810.txt docID=89 score=0.711262
Press (n)ext page, (q)uit or enter number to jump to a page.
n
PATH : docs\ 822.txt docID=101 score=0.7022961
PATH : docs\ 735.txt docID=14 score=0.68991125
PATH : docs\ 739.txt docID=18 score=0.6651991
PATH : docs\ 811.txt docID=90 score=0.6561749
PATH : docs\ 762.txt docID=41 score=0.6421961
PATH : docs\ 785.txt docID=64 score=0.6421961
PATH : docs\ 779.txt docID=58 score=0.631281
PATH : docs\ 751.txt docID=30 score=0.6105275
PATH : docs\ 778.txt docID=57 score=0.60213274
ANTLR Preview
Tool Output
9: Version Control
Terminal
4: Run
6: TODO
Event Log
All files are up-to-date (a minute ago)
Material Oceanic
43:1 CRLF UTF-8 Git: main
ENG
12:53 PM
10/18/2020


IR_HW3_LUCENE_NEW [F:\info retrieval\homework\IR_HW3_LUCENE_NEW] - ...\SearchFiles.java [IR_HW3_LUCENE_NEW] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
PATH : docs\ 822.txt docID=101 score=0.7022961
PATH : docs\ 735.txt docID=14 score=0.68991125
PATH : docs\ 739.txt docID=18 score=0.6651991
PATH : docs\ 811.txt docID=90 score=0.6561749
PATH : docs\ 762.txt docID=41 score=0.6421961
PATH : docs\ 785.txt docID=64 score=0.6421961
PATH : docs\ 779.txt docID=58 score=0.631281
PATH : docs\ 751.txt docID=30 score=0.6105275
PATH : docs\ 778.txt docID=57 score=0.60213274
PATH : docs\ 782.txt docID=61 score=0.5910951
Press (p)revious page, (n)ext page, (q)uit or enter number to jump to a page.
2
PATH : docs\ 822.txt docID=101 score=0.7022961
PATH : docs\ 735.txt docID=14 score=0.68991125
PATH : docs\ 739.txt docID=18 score=0.6651991
PATH : docs\ 811.txt docID=90 score=0.6561749
PATH : docs\ 762.txt docID=41 score=0.6421961
PATH : docs\ 785.txt docID=64 score=0.6421961
PATH : docs\ 779.txt docID=58 score=0.631281
PATH : docs\ 751.txt docID=30 score=0.6105275
PATH : docs\ 778.txt docID=57 score=0.60213274
PATH : docs\ 782.txt docID=61 score=0.5910951
Press (p)revious page, (n)ext page, (q)uit or enter number to jump to a page.
q
Enter query:
ANTLR Preview
Tool Output
9: Version Control
Terminal
4: Run
6: TODO
Event Log
All files are up-to-date (a minute ago)
Material Oceanic
43:1 CRLF UTF-8 Git: main
ENG
12:54 PM
10/18/2020